Programming Models & Runtime Systems

Breakout Report

MICS PI Meeting, June 27, 2002

Question 1

What are the unfunded (and/or) under-funded) basic research issues critical to our ability to exploit next-generation systems?

Programming Models for 1M threads

- Most common programming languages are based on linear execution of a single thread; SPMD is an implicit array of such single threads. It is hard to imagine this can bring us to 1M threads (herding weasels). We need imaginative research on programming models that are based on different abstractions (no variables?, no sequential execution?)

Locality Handling

- Current programming models and languages have no good representation for locality; languages focus on control and computation, not on data location and data movement. What are ways of providing different levels of abstraction for data location, data movement, and locality? What is the equivalent of procedural abstraction that we have for computation?

Programmer Productivity

- Higher level programming languages
  - Bring Mathematica/Matlab like environments to HPC
  - Languages that support very fast turnaround on relatively small (but computationally heavy) codes, to enable fast experimentation, or fast time to solution.
  - Support the ability to later refine it into tuned code and to integrate it into larger systems
    - Rapid prototyping for HPC

Fault Tolerance

- Should it surface to runtime and/or programming model?
- What is the right combination of hw/system/runtime/PL?
- How do we avoid global checkpoints? How do we support high performance computing reactive environments?
- Trustworthy computing in general: solutions to software bugs, intruder attacks…

On the Fly Code Tuning

- Scientific codes run for long times. By collecting data during execution one can continuously tune the code. This requires the right language/compiler/runtime environment.
  - Same idea applies a fortiori to libraries

Flexible Run-Time

- Suppose that the architecture provides finer grain, more fluid “objects” (threads, data objects), to facilitate migration, load balancing, fault isolation. How do we build PLs and runtime to take advantage of it?

Compiler Infrastructure

- Research in parallelizing compilers and compilation for scientific computing is shrinking. Vendors are disinvesting. Should we invest in common compiler infrastructure? (e.g., front ends)
- How do we develop more modular compilation approaches, to support plugin compiler components?
- Compiler toolkits to facilitate research sharing
- Locality aware code generation techniques
- Compilers that interact with user to choose optimizations, and are actively used for performance analysis

Semantically Rich Interfaces

- Find ways of conveying more information at library interfaces – e.g., semantic information that can be provided by user, or deduced by compiler, or performance information, or common use information. This e.g. to enable separate optimization of library invocations and, in general, modular optimization.

Question 2

What do you need most from the other four CS research groups to enable your future progress and enhance the effectiveness of your work?

Lots of Help

- Viz/data: Large scale performance/behavior visualization
  - Current tools scale badly
  - Discrete data visualization as opposed to continuous fields
  - Relate program, performance data, compiler decisions…
- Interoperability/portability – CCA ideas could be expanded to provide semantic information across interfaces, including performance information to support global resource management
  - Ways to express component semantics so as to support composition
  - CCA component ideas applied to infrastructure components as well as application components: e.g. compilers, debuggers

Even More Help

- Performance: Real time performance measurements that support feedback directed optimizations. Hardware/system support for performance information aggregation
- OS:
  - An OS that does not require OS bypass: bypass is needed because OS does not have right interface/function/performance. Develop virtual machines with right mechanisms
  - Hierarchical and dynamic resource management; ability to dynamically allocate all system resources, to grow and shrink jobs; interface for dialogue and negotiation between system and runtime for resource allocation (including distributed systems); two-level allocation, to job and within job for all resources; application specific policies.
  - Fast job startup/rundown
  - Support for fault tolerance, e.g. virtualization and remapping of all resources. Also support for fault tolerance by application.

Question 3

What are the current gaps in the MICS CS research portfolio related to peta-scale computing? What new topic areas should be added to the MICS portfolio?

Other things that would help but aren’t being addressed

- Formal methods for verification, static and dynamic checking; assertion languages, dynamic invariant checking, introspection…
- Debugging (relates to formal methods)
- Compiler support for HPC
- Architecture research
  - Funding for new programs or for tie ins with existing projects (DARPA funding)
- Computing facilities for simulations (large scale), new hardware testing facilities (small scale toy shop)

The End